


```
CCCCCCCC  RRRRRRRR  FFFFFFFFFF  MM      MM  EEEEEEEEE  MM      MM
CCCCCCCC  RRRRRRRR  FFFFFFFFFF  MM      MM  EEEEEEEEE  MM      MM
CC         RR      RR  FF      FF  MMMM  MMMM  EE      EE  MMMM  MMMM
CC         RR      RR  FF      FF  MMMM  MMMM  EE      EE  MMMM  MMMM
CC         RR      RR  FF      FF  MM      MM  EE      EE  MM      MM
CC         RRRRRRRR  FFFFFFFF  MM      MM  EEEEEEE  MM      MM
CC         RRRRRRRR  FFFFFFFF  MM      MM  EEEEEEE  MM      MM
CC         RR      RR  FF      FF  MM      MM  EE      EE  MM      MM
CC         RR      RR  FF      FF  MM      MM  EE      EE  MM      MM
CC         RR      RR  FF      FF  MM      MM  EE      EE  MM      MM
CC         RR      RR  FF      FF  MM      MM  EE      EE  MM      MM
CCCCCCCC  RR      RR  FF      FF  MM      MM  EEEEEEEEE  MM      MM
CCCCCCCC  RR      RR  FF      FF  MM      MM  EEEEEEEEE  MM      MM
                                     ....
                                     ....
                                     ....
                                     ....
```

```
LL          IIIIII  SSSSSSSS
LL          IIIIII  SSSSSSSS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SSSSSS
LL          II      SSSSSS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SS
LLLLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLLLL  IIIIII  SSSSSSSS
```


GETMEM
Table of contents

Allocate/deallocate virtual memory ^E 9

15-SEP-1984 23:37:57 VAX/VMS Macro V04-00

Page 0

(2) 56
(3) 75
(4) 191

declarations

get_mem

FREE_MEM

Allocate memory

Free virtual memory

Handwritten signature

```
0000 1 .TITLE GETMEM Allocate/deallocate virtual memory
0000 2 .IDENT 'V04-000'
0000 3
0000 4
0000 5 *****
0000 6 *
0000 7 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 8 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9 * ALL RIGHTS RESERVED.
0000 10 *
0000 11 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 12 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 13 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 14 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 15 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 16 * TRANSFERRED.
0000 17 *
0000 18 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 19 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 20 * CORPORATION.
0000 21 *
0000 22 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 23 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 24 *
0000 25 *
0000 26 *****
0000 27
0000 28
0000 29 ++
0000 30 FACILITY: Memory allocation routines
0000 31
0000 32 ABSTRACT: ALLOCATE AND DEALLOCATE VIRTUAL BLOCK
0000 33
0000 34
0000 35 ENVIRONMENT: VAX Native mode
0000 36
0000 37 AUTHOR: K.D. MORSE, CREATION DATE: 25-APR-77
0000 38
0000 39 MODIFIED BY:
0000 40
0000 41 V03-001 JWT0032 Jim Teague 25-May-1982
0000 42 Allow larger vm requests for crfs.
0000 43
0000 44 X01.01 001 B.L. SCHREIBER 9-FEB-1979
0000 45 Correct error in allocation routine.
0000 46 V01.02 008 B.L. SCHREIBER 26-OCT-1979
0000 47 Declare $CRFMSG
0000 48 V01.03 010 B.L. SCHREIBER 13-NOV-1979
0000 49 Word-relative references
0000 50 V01.04 B.L. SCHREIBER 15-NOV-1979
0000 51 Generalize into GETMEM.
0000 52 V01.05 B.L. SCHREIBER 22-JAN-1980
0000 53 Round request up to 8 bytes.
0000 54 --
```



```

0000 56      .SBTTL declarations
0000 57      :
0000 58      : Declare macros
0000 59      :
0000 60      $crfdef      ; cref control table offsets
0000 61      $libdef      ; rtl error message defs
0000 62      :
0000 63      : EQUATED SYMBOLS:
0000 64      :
0000 65      :
00000000 0000 66 blk$_addr = 0      ; Offset to addr of next block
00000004 0000 67 blk$_size = 4      ; Offset to size of this block
00000008 0000 68 mem$_roundup = 8    ; Round up request size
00158264 0000 69 mem$_blkwithinbl = lib$_badbloadr ; Block deallocated within deallocat
0015826C 0000 70 mem$_illblksize = lib$_badblosiz  ; Illegal block size
00000001 0000 71 mem$_success = 1    ; Success
0000 72      :
0000 73      .default displacement, word ; Use word displacement

```

```
0000 75 .SBTTL get_mem Allocate memory
0000 76 :++
0000 77 : Functional description:
0000 78 :
0000 79 : This routine allocates a block of dynamic memory. The requested block
0000 80 : size is rounded up to the nearest eight bytes. An error condition is
0000 81 : returned if the block cannot be allocated.
0000 82 : First fit algorithm is used.
0000 83 :
0000 84 : Calling sequence:
0000 85 :
0000 86 : BSBW get_mem
0000 87 :
0000 88 : Input parameters:
0000 89 :
0000 90 : R0 Address of longword containing number of bytes to allocate
0000 91 : R1 Address of longword to store allocated memory address
0000 92 : R11 Pointer to cross reference control table
0000 93 :
0000 94 : Completion codes:
0000 95 :
0000 96 : success:
0000 97 : r0 - contains a one
0000 98 : failure:
0000 99 : r0 - contains a zero
0000 100 :
0000 101 : Side effects:
0000 102 :
0000 103 : The dynamic memory list is updated. More dynamic memory is acquired
0000 104 : if necessary
0000 105 :
0000 106 :
0000 107 :--
0000 108 :
0000 109 :
0000 110 .PSECT $CODE$, NOPIC, USR, CON, REL, LCL, NOSHR, EXE, RD, NOWRT, NOVEC
0000 111
0000 112 get_zmem::
0000 113 pushr #^m<r0,r1> ; save r0/r1
0000 114 bsbb get_mem ; allocate the memory
0000 115 blbc r0,T0$ ; branch if no memory today
0000 116 popr #^m<r0,r1> ; restore arguments
0000 117 pushr #^m<r2,r3,r4,r5> ; save registers over movc
0000 118 movc5 #0,(sp),#0,r0,@(r1) ; zero memory
0000 119 popr #^m<r2,r3,r4,r5> ; restore registers
0000 120 movl #mem$_success,r0
0000 121 rsb
0000 122 10$: cmpl (sp)+,(sp)+ ; clear saved r0/r1 from stack
0000 123 rsb ; return with error in r0
0000 124
0000 125 crf$get_mem::
0000 126 pushl r11 ; save r11
0000 127 movl r2,r11 ; set control table addr into r11
0000 128 bsbb get_mem ; allocate the memory
0000 129 popl r11 ; restore r11
0000 130 rsb
0000 131
```

00 B1 50 00 6E 00 2C 000B 118
11 50 01 D0 0014 120
8E 8E D1 0018 122
05 0018 123
001C 124
001C 125
5B 5B DD 001C 126
5B 52 D0 001E 127
04 10 0021 128
5B 8E D0 0023 129
05 0026 130
0027 131


```
0027 132 get_mem::
0027 133      pushr    #^M<r2, r3, r4>          ; Save registers
0029 134      movl    r1,r4                    ; Save return result address
002C 135      addl3   #<mem$c_roundup-1>,r0,r3 ; Round up to the nearest
0030 136      bicl2   #<mem$c_roundup-1>,r3    ; Multiple of eight bytes
0033 137      bleq    10$                    ; Check for size <= 0
0035 138      cmpl    r3,crf$l_maxblk(r11)    ; Check size > maximum
0039 139      bleq    40$                    ; Branch on legal size
003B 140      movl    r3,crf$l_maxblk(r11)    ; If > maximum, store # bytes
003F 141      ashl    #-9,r3,crf$l_memexp(r11) ; and page count
0045 142      incl    crf$l_memexp(r11)      ; Add one page for excess bytes
0048 143      brb     40$                    ; Go try to allocate it
004A 144 10$:      movl    #mem$_illblksize,r0 ; Report illegal block size
0051 145      brb     al_blk_exit            ; Return
0053 146      ;
0053 147      ; Expand the program region
0053 148      ;
0053 149 20$:      clrq    -(sp)                ; Make room on stack for return
0055 150      movl    sp,r1                  ; copy the address
0058 151      $EXPREG_S      PAGCNT=crf$l_memexp(r11),- ; Expand dynamic memory
0058 152      RETADR=(r1)
0068 153      movq    (sp)+,r1                ; get return info from stack
006B 154      blbc    r0,al_blk_exit          ; Branch if it failed
006E 155      ;
006E 156      ; Now insert memory into list.
006E 157      ;
006E 158 30$:      movl    crf$l_maxblk(r11),r0 ; Get size of new block
0072 159      bsbb    free_mem              ; Go insert new blk in list
0074 160      blbc    r0,al_blk_exit          ; Branch if failed to insert new blo
0077 161      ;
0077 162      ; Search down list for first block >= size requested.
0077 163      ;
0077 164 40$:      movab    crf$l_dynmem(r11),r1 ; Get listhead of dynamic memory
007B 165 50$:      movl    r1,r2              ; Set new previous block
007E 166      movl    (r2),r1                ; Get address of free block
0081 167      beql    20$                    ; End of list, go expand memory
0083 168      cmpl    r3,blk$l_size(r1)      ; Requested size > block size?
0087 169      bgtr    50$                    ; Yes, keep looking
0089 170      beql    60$                    ; Branch on same size
008B 171      ;
008B 172      ; Take part of this block and link the rest back into the list.
008B 173      ;
008B 174      subl2    r3,blk$l_size(r1)      ; Subtract off requested size
008F 175      cmpl    blk$l_size(r1),#8      ; Did we allocate the bookkeeping wo
0093 176      bleq    60$                    ; If so, go pretend it's a good fit.
0095 177      ; This can float at most 4 bytes.
0095 178      addl2    blk$l_size(r1),r1      ; Get address of requested block
0099 179      brb     70$                    ; Return
009B 180      ;
009B 181      ; Block was perfect fit. delete it from the list.
009B 182      ;
009B 183 60$:      movl    blk$l_addr(r1),blk$l_addr(r2) ; Set pointer to next block
009E 184      clrq    blk$l_addr(r1)        ; Clean up the block
00A0 185 70$:      movl    #mem$_success,r0  ; Set success status code
00A3 186      movl    r1,(r4)                ; Return address to caller
00A6 187 al_blk_exit:
00A6 188      popr     #^M<r2, r3, r4>
```

GETMEM
V04-000

Allocate/deallocate virtual memory J 9
get_mem Allocate memory

15-SEP-1984 23:37:57 VAX/VMS Macro V04-00
4-SEP-1984 23:38:56 [CRF.SRC]CRFMEM.MAR;1

Page 5
(3)

05 00A8 189 rsb


```
00A9 191 .SBTTL FREE_MEM Free virtual memory
00A9 192 :++
00A9 193 : Functional description:
00A9 194 :
00A9 195 : This routine deallocates a block of memory and inserts it into a dynamic
00A9 196 : Memory list. The block is zeroed and its size is rounded up to the
00A9 197 : Nearest eight bytes. If it is adjacent to another block, the two blocks
00A9 198 : are compacted into one.
00A9 199 :
00A9 200 : Calling sequence:
00A9 201 :
00A9 202 : BSBW FREE_MEM
00A9 203 :
00A9 204 : Input parameters:
00A9 205 :
00A9 206 : R0 size of block to deallocate
00A9 207 : R1 address of block to deallocate
00A9 208 : R11 Pointer to cross reference control block
00A9 209 :
00A9 210 : Completion codes:
00A9 211 :
00A9 212 : success:
00A9 213 : r0 - contains a one
00A9 214 : failure:
00A9 215 : r0 - contains a zero
00A9 216 :
00A9 217 : Side effects:
00A9 218 :
00A9 219 : NONE
00A9 220 :
00A9 221 :--
00A9 222 :
00A9 223 :
00A9 224 crf$free_mem::
00A9 225 free_mem::
00A9 226 pushr #^m<r2, r3, r4>
00AB 227 addl3 #<mem$c_roundup-1>,r0,r3 : Round up to the nearest
53 50 07 C1 00AF 228 10$: bicl2 #<mem$c_roundup-1>,r3 : Multiple of eight bytes
53 53 07 CA 00B2 229 bleq 20$ : Check for size <= 0
18 AB 06 15 00B4 230 cmpl r3,crf$l_maxblk(r11) : Check size > maximum
50 0015826C 8F D0 00B8 231 bleq 30$ : Branch on legal size
50 0062 31 00C1 232 20$: movl #mem$_illblksize,r0 : Report illegal block size
50 10 AB 9E 00C4 233 brw deal_blk_exit : Return
54 53 61 7C 00C8 234 30$: movab crf$l_dynmem(r11),r0 : Get listhead of dynamic memory
54 53 51 C1 00CA 235 clrq (r1) : Clear block info long words
54 53 51 C1 00CE 236 addl3 r1,r3,r4 : Get address following new block
00CE 237 :
00CE 238 : Search down list for insertion point.
00CE 239 :
52 50 D0 00CE 240 40$: movl r0,r2 : R2 contains prev block
50 60 D0 00D1 241 movl (r0),r0 : R0 contains next block addr
50 25 13 00D4 242 beqlu 60$ : Branch on end of list
50 51 D1 00D6 243 cmpl r1,r0 : Is new addr > next addr
50 51 D1 00D9 244 bgtru 40$ : Yes, keep looking
00DB 245 :
00DB 246 : Found insertion point.
00DB 247 : r2 = addr of previous block
```

```
00DB 248 : r1 = addr of new block
00DB 249 : r0 = addr of next block
00DB 250 :
04 62 51 D0 00DB 251 : movl r1,blk$l_addr(r2) ; Point prev to new
50 54 D1 00DE 252 : cmpl r4,r0 ; Is new adjacent to next?
0B 13 00E1 253 : beqlu 50$ ; Yes, branch to compact
35 1A 00E3 254 : bgtru 80$ ; Error, within block
04 61 50 D0 00E5 255 : movl r0,blk$l_addr(r1) ; No, point new to next
A1 53 D0 00E8 256 : movl r3,blk$l_size(r1) ; Set size of new block
14 11 00EC 257 : brb 70$ ; Go check adjacent to prev
00EE 258 :
00EE 259 : Compact with next block.
00EE 260 :
04 A1 04 A0 53 C1 00EE 261 50$: addl3 r3,blk$l_size(r0),blk$l_size(r1) ; Set size = new+next
61 60 D0 00F4 262 : movl blk$l_addr(r0),blk$l_addr(r1) ; Set next pointer
60 7C 00F7 263 : clrq (r0) ; Clear old next block
07 11 00F9 264 : brb 70$ ; Go check for compaction
00FB 265 :
00FB 266 : Set up new block on end of list.
00FB 267 :
04 62 51 D0 00FB 268 60$: movl r1,blk$l_addr(r2) ; Point prev to new
A1 53 D0 00FE 269 : movl r3,blk$l_size(r1) ; Set new block size
0102 270 :
0102 271 : Check for compact with previous block.
0102 272 :
54 04 A2 52 C1 0102 273 70$: addl3 r2,blk$l_size(r2),r4 ; Get end of prev block
51 54 D1 0107 274 : cmpl r4,r1 ; Is new adjacent to prev?
0E 1A 010A 275 : bgtru 80$ ; Error, block within prev
15 12 010C 276 : bnequ 90$ ; No, not adjacent
010E 277 :
010E 278 : Compact with previous block.
010E 279 :
04 A2 04 A1 C0 010E 280 : addl2 blk$l_size(r1),blk$l_size(r2) ; Prev size = new+prev
62 61 D0 0113 281 : movl blk$l_addr(r1),blk$l_addr(r2) ; Set up next pntr
61 7C 0116 282 : clrq (r1) ; Clear new block pntr & size
09 11 0118 283 : brb 90$ ; Branch to exit
011A 284 :
011A 285 : Error, block within block.
011A 286 :
50 00158264 8F D0 011A 287 80$: movl #mem$ blkwithinbl,r0 ; Report block within block
03 11 0121 288 : brb deal_blk_exit ; Return
50 01 D0 0123 289 90$: movl #mem$_success,r0 ; Set success status code
0126 290 : deal_blk_exit:
1C BA 0126 291 : popr #^m<r2, r3, r4>
05 05 0128 292 : rsb
0129 293 :
0129 294 : .end
```


GETMEM
Symbol table

Allocate/deallocate virtual memory M 9

15-SEP-1984 23:37:57 VAX/VMS Macro V04-00
4-SEP-1984 23:38:56 [CRF.SRC]CRFMEM.MAR;1

Page 8
(4)

```
$ST1      = 00000000
AL_BLK_EXIT  = 000000A6 R      02
BLKSL_ADDR  = 00000000
BLKSL_SIZE  = 00000004
CRFSFREE_MEM = 000000A9 RG     02
CRFSGET_MEM  = 0000001C RG     02
CRFSL_DYNMEM = 00000010
CRFSL_MAXBLK = 00000018
CRFSL_MEMEXP = 0000001C
DEAL_BLK_EXIT = 00000126 R      02
FREE_MEM     = 000000A9 RG     02
GET_MEM      = 00000027 RG     02
GET_ZMEM     = 00000000 RG     02
LIBS_BADBLOADR = 00158264
LIBS_BADBLOSIZ = 0015826C
MEMSC_ROUNDUP = 00000008
MEMS_BLKWITHINBL = 00158264
MEMS_ILBLKSIZE = 0015826C
MEMS_SUCCESS  = 00000001
SYSS$EXPREG   ***** GX     02
```

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes
ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$ABSS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
\$CODE\$	00000129 (297.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC BYTE

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	35	00:00:00.05	00:00:00.42
Command processing	138	00:00:00.62	00:00:03.57
Pass 1	160	00:00:02.48	00:00:06.87
Symbol table sort	0	00:00:00.17	00:00:01.11
Pass 2	68	00:00:00.78	00:00:01.28
Symbol table output	4	00:00:00.03	00:00:00.06
Psect synopsis output	1	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	408	00:00:04.15	00:00:13.34

The working set limit was 1050 pages.
12559 bytes (25 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 142 non-local and 17 local symbols.
294 source lines were read in Pass 1, producing 13 object records in Pass 2.
12 pages of virtual memory were used to define 11 macros.

! Macro library statistics !

Macro library name

Macros defined

\$255\$DUA28:[CRF.OBJ]CRF.MLB;1
\$255\$DUA28:[SYSLIB]STARLET.MLB;2
TOTALS (all libraries)

1
7
8

307 GETS were required to define 8 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:CRFMEM/OBJ=OBJ\$:CRFMEM MSRC\$:CRFMEM/UPDATE=(ENH\$:CRFMEM)+LIB\$:CRF/LIB

0068 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY